A.4 Python Exceptions Reference

Adapted from https://docs.python.org/3/library/exceptions.html. Note: Not all built-in Python exceptions are shown.

Exception	Description
`AssertionError`	Raised when an `assert` statement fails.
`AttributeError`	Raised when an attribute reference or assignment fails. (When an object does not support attribute references or attribute assignments at all, `TypeError` is raised.)
`FileNotFoundError`	Raised when a file or directory is requested but doesn't exist.
`ImportError`	Raised when the import statement has troubles trying to load a module. Also raised when the "from list" in `from ... import` has a name that cannot be found.
`ModuleNotFoundError`	A subclass of `ImportError` which is raised by `import` when a module could not be located.
`IndexError`	Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not an integer, `TypeError` is raised.)
`KeyError`	Raised when a mapping (dictionary) key is not found in the set of existing keys.
`NameError`	Raised when a local or global name is not found.
`NotImplementedError`	In user defined base classes, abstract methods should raise this exception when they require derived classes to override the method, or while the class is being developed to indicate that the real implementation still needs to be added.
`RecursionError`	It is raised when the interpreter detects that the maximum recursion depth (see `sys.getrecursionlimit()`) is exceeded.
`SyntaxError`	Raised when the parser encounters a syntax error. This may occur in an import statement, in a call to the built-in functions `exec()` or `eval()`, or when reading the initial script or standard input (also interactively).
`IndentationError`	Base class for syntax errors related to incorrect indentation.
`TabError`	Raised when indentation contains an inconsistent use of tabs and spaces.

Exception	Description
`TypeError`	Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch. This exception may be raised by user code to indicate that an attempted operation on an object is not supported, and is not meant to be. If an object is meant to support a given operation but has not yet provided an implementation, `NotImplementedError` is the proper exception to raise. Passing arguments of the wrong type (e.g. passing a list when an int is expected) should result in a `TypeError`, but passing arguments with the wrong value (e.g. a number outside expected boundaries) should result in a `ValueError`.
`ValueError`	Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as `IndexError`.
`ZeroDivisionError`	Raised when the second argument of a division or modulo operation is zero. The associated value is a string indicating the type of the operands and the operation.